

Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Виноградов Никита Андреевич

**ОРГАНИЗАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ.
СТРУКТУРНЫЙ ПАТТЕРН ЗАМЕСТИТЕЛЬ**

Лабораторная работа

студента образовательной программы «Программная инженерия»
по направлению подготовки 09.03.04 Программная инженерия

Руководитель
к.т.н., доцент кафедры Информационных технологий в бизнесе НИУ ВШЭ-Пермь

А.В. Кычкин

Пермь, 2020 год

Оглавление

Глава 1. Заместитель	3
1.1 Назначение	3
1.2 Структура	3
1.3 Способ применения	4
Глава 2. Реализация паттернов	5
2.1 Диаграмма классов	5
2.2 Диаграмма последовательности	6
2.3 Код программы	6

Глава 1. Заместитель

это структурный паттерн проектирования, который позволяет подставлять вместо реальных объектов специальные объекты-заменители. Эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу.

1.1. Назначение

Заместитель предлагает создать новый класс-дублёр, имеющий тот же интерфейс, что и оригинальный служебный объект. При получении запроса от клиента объект-заместитель сам бы создавал экземпляр служебного объекта и переадресовывал бы ему всю реальную работу.

1.2. Структура

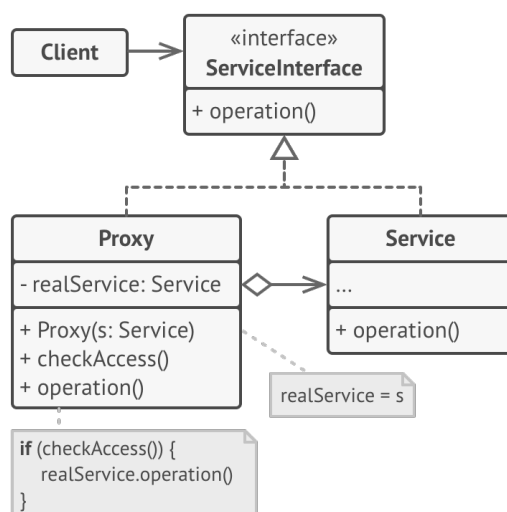


Рис. 1.1. Структура классов паттерна Заместитель

Участники:

- *Client* - класс который вызывает методы компонента
- *Service Interface* - определяет общий интерфейс для сервиса и заместителя
- *Proxy* - хранит ссылку на объект сервиса , передаёт вызовы вложенному сервису.

- *Service* - Класс содержащий бизнес-логику проекта

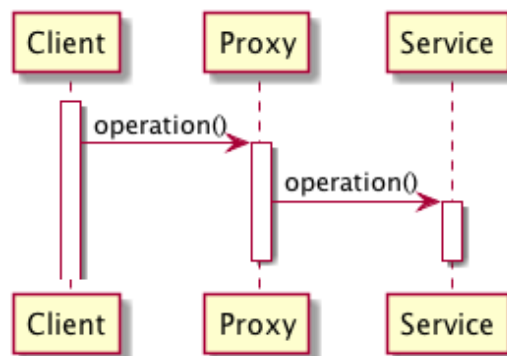


Рис. 1.2. Диаграмма последовательности паттерна Заместитель

1.3. Способ применения

Данный паттерн необходим когда необходимо перехватывать операции в системе для валидации данных или придания дополнительной логики.

Глава 2. Реализация паттернов

2.1. Диаграмма классов

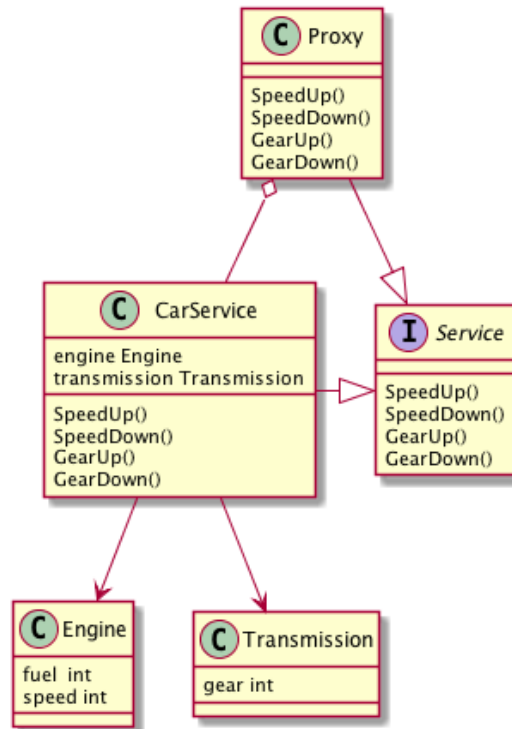


Рис. 2.1. Диаграмма классов паттерна Компоновщик

Участники:

- *Client* - класс который вызывает методы компонента
- *Service Interface* - определяет общий интерфейс для сервиса и заместителя
- *Proxy* - хранит ссылку на объект сервиса , передаёт вызовы вложенному сервису.
- *CarService* - Сервис вызывающий функции в элементах автомобиля

2.2. Диаграмма последовательности

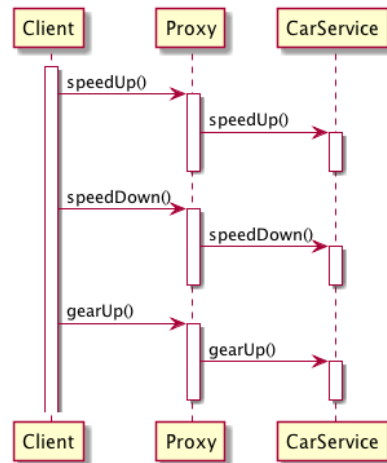


Рис. 2.2. Диаграмма последовательности паттерна Компонировщик

2.3. Код программы

```
package car
```

```
import (  
    "container/list"  
    "reflect"  
)
```

```
func TypeFlex(v interface{}) string {  
    return reflect.TypeOf(v).String()  
}
```

```
type Car struct {  
    Container *list.List  
}
```

```
func (e Car) Print(num int) {  
    for e := e.Container.Front(); e != nil; e = e.Next() {  
  
        switch str := e.Value.(type) {
```

```

case CartPart:
println(TypeFlex(e.Value))

str.Print(num + 1)
break
}
}
}
func (e Car) AddPart(part interface{}) {

e.Container.PushBack(part)
println(e.Container.Len())
}
func (e Car) RemovePart(part *interface{}) {
el := list.Element{Value: part}
e.Container.Remove(&el)
}


type CartPart interface {
Print(num int)
}
package car


import "container/list"


//Демпфирующий
//Стабилизатор
//Стойка
//Рычаг
//Пружина
//Амортизатор


type Damping struct {

```

```

}
type Stabilizer struct {
}
type Rack struct {
}
type Lever struct {
}
type Spring struct {
}
type Absorber struct {
}
type Suspension struct {
    Container *list.List
}

func (e Suspension) Print(num int) {
for e := e.Container.Front(); e != nil; e = e.Next() {
for i := 0; i < num; i++ {
print("—")
}
println(TypeFlex(e.Value))
}
}

func (e Suspension) AddPart(part interface{}) {
    e.Container.PushBack(part)
}

func (e Suspension) RemovePart(part *interface{}) {
    el := list.Element{Value: part}
    e.Container.Remove(&el)
}

func NewSuspension() Suspension {

```



```

return Suspension{Container: list.New()}
}

package car

//блока цилиндровкартером ,
//
//- головкиблокацилиндров ,
//
//- поддонакартерадвигателя ,
//
//- поршнейскольцамиипальцами ,
//
//- шатунов ,
//
//- коленчатоговала ,
//
//- маховика.
import (
    "container/list"
)

type Engine struct {
    Container *list.List
}

func (e Engine) Print(num int) {
    for e := e.Container.Front(); e != nil; e = e.Next() {
        for i := 0; i < num; i++ {
            print(" ")
        }
        println(TypeFlex(e.Value))
    }
}

```

```

type CylinderBlock struct {
}
type Sump struct {
}
type Crankshaft struct {
}
type Rod struct {
}

func (e Engine) AddPart(part interface{}) {
e.Container.PushBack(part)
}
func (e Engine) RemovePart(part *interface{}) {
el := list.Element{Value: part}
e.Container.Remove(&el)
}

type EnginePart struct {
}

```